Assignment 3 – CNNs for Image Classification (CIFAR-10)

Course: CMPUT 328 — Visual Recognition
Deadline: October 5, 11:59 pm
Total Weight: 5% of final grade
Parts: A. In-lab (3 hrs), **B. Take-home (1 week)**

Overview

You will build a small convolutional neural network (CNN) in PyTorch to classify images from **CIFAR-10**. In-lab you will implement and train a minimal working model on a **subset** of the data to verify your pipeline. At home you will scale up to the **full dataset**, add **data augmentation**, and compare results to the **fully connected NN (Assignment 2)**.

Key skills: model architecture, convolutions/activations/pooling, data loaders, augmentation, training loops, metrics, and experiment reporting.

Learning goals

By the end of this assignment, you should be able to:

- Explain why CNNs do **not** flatten the image at the input and how spatial structure is preserved layer-to-layer.
- Implement a compact CNN (Conv→ReLU→Pool stacks; optional BatchNorm/Dropout) and train it on CIFAR-10.
- Use common **augmentations** (random crop/flip/colour jitter/cutout) and discuss their impact on generalization.
- Run a **fair comparison** against a fully connected baseline (from Assignment 2) using matched training setups and report metrics clearly.

Dataset & environment

- **Dataset:** CIFAR-10 (60k colour images, 32×32, 10 classes). Use torchvision.datasets.CIFAR10 with train=True/False.
- **Device:** GPU if available (Colab or local). Your code must still run on CPU.
- **Reproducibility:** set seeds (e.g., torch.manual_seed(328), random.seed(328), np.random.seed(328)).

Part A — In-lab (2% of course grade)

Goal: Stand up an end-to-end training pipeline and reach working accuracy on a small subset.

A.1 Tasks

1. **Implement the model** (example starter below). Keep it small enough to train quickly.
2. **Dataset:** use **45,000 training images** (balanced by class if you wish) and the **full 10,000 test** or a **5,000-image validation** split. Document your split.
3. **Training:** 5–10 epochs, Adam or SGD, reasonable LR (e.g., 1e-3 with Adam). Track **train/val loss & accuracy** each epoch.
4. **Sanity checks:**
    - Overfit to a tiny batch (e.g., 128 images) to confirm the training loop works.
    - Verify tensor shapes after each layer (no flatten at input!).
5. **Save artifacts:** final metrics, loss/accuracy curves, and a few example predictions.

A.2 Minimal CNN starter (illustrative)

Conv(3→32, k=3, padding=1) → ReLU → MaxPool(2)

Conv(32→64, k=3, padding=1) → ReLU → MaxPool(2)

Conv(64→128, k=3, padding=1) → ReLU → GlobalAvgPool

Linear(128→10)

Notes: You may add BatchNorm/Dropout after convs; keep parameter count modest for speed.

A.3 Expected outcome (subset)

- **Baseline target:** ≥ **45–55%** test accuracy on the subset after ~10 epochs is reasonable. Record exact settings to reproduce.

A.4 Save and load model checkpoints (NOT graded, but recommended)

You should save your model checkpoints and learn how to load and test it.

Part B — Take-home (3% of course grade)

Goal: Train on **full CIFAR-10** with **augmentations**, tune modestly, and compare to your **Assignment 2 fully-connected NN**.

B.1 Required augmentations (choose ≥2)

- RandomCrop(32, padding=4)
- RandomHorizontalFlip()
- ColorJitter (mild)
- RandomErasing / Cutout (small holes)
  Describe each choice and its expected effect.

B.2 Fine-tuning

- **Epochs:** 30–60 (pick and justify); **Optimizer:** Adam or SGD+Momentum.
- **LR schedule:** optional StepLR or CosineAnnealing. State settings.
- **Regularization:** optional weight decay (e.g., 5e-4). Track training time & best epoch.
- **Validation:** hold out 5k images from training as validation (or use test set only for final report—be explicit and consistent).

B.3 Fair comparison to Assignment2 (fully connected)

- Re-run (or reuse) your Assignment2 with the **same data split** and **similar epoch budget**.
- Report **accuracy**, **loss**, and at least one **calibration/uncertainty** proxy (e.g., confidence histograms or softmax max-prob distribution).
- Briefly discuss: *Why does CNN outperform (or not)?* Your response should tie to spatial bias, parameter efficiency, and augmentation.

B.4 Accuracy bands (full data, small CNN)

- **Not ideal:** Under 65% → **0% of your result grade for Part B**
- **Solid pass:** 65–70% → **50% of your result grade for Part B**
- **Good:** 70–80% → **75% of your result grade for Part B**
- **Strong:** 80–85%+ → **100% of your result grade for Part B**

  Document your best accuracy.

Deliverables checklist

- Part A subset run with curves & quick notes
- Part B full run with ≥2 augmentations
- Comparison table vs Assignment2
- 3-page-max PDF report with plots & examples
- Repro instructions + seeds

What to hand in (submission package)

Submit a single compressed folder **A3_CNN_<CCID>.zip** to canvas containing:

1. **Notebook:** Assignment3_Template.ipynb (must run top-to-bottom on fresh runtime). Add a **requirements.txt** if non-standard packages are used.
2. **Report (max 3 pages, PDF):** concise narrative with the following sections:
    - **Methods:** architecture diagram or table, hyperparameters, augmentations.
    - **Results:** accuracy/loss curves, final metrics (with validation/test split clearly stated), a 3×3 grid of example predictions with correct/incorrect cases.
    - **Comparison to Assignment 2:** table + 1–2 paragraphs of analysis.
    - **Ablation (short):** effect of removing augmentations or BatchNorm/Dropout.
3. **Code artifacts:** model file(s) if you saved checkpoints (.pt) — optional.

Marking breakdown (5% total)

Part A (In-lab) — 2%

Section	Requirements	Value (%)	Notes (Expectations for Excellent)
1	Pipeline working on subset, with logs/curves	1.0	Correct CNN, tidy modules, verified shapes; stable training
2	Architecture correct (no flatten at input), shapes verified; basic accuracy target met	0.8	Verified architecture, no major flaws, meets accuracy baseline
3	Clean code & brief in-lab notes	0.2	Clean, well-organized code; clear and concise notes
Total		**2.0**	

Part B (Take-home) — 3%

	Requirements	Value (%)	Notes (Expectations for Excellent)
1	Full-data training + ≥2 augmentations, settings documented	1.0	≥3 well-justified augmentations with impact quantified
2	Results quality (reasonable band) + clear plots/tables	1.0	Strong, consistent metrics; clean curves and insightful examples Your score of this section will depend on your best model test accuracy, as listed in description
3	Fair comparison to Assignment 2 with thoughtful discussion	0.7	Rigorous, fair, well-argued comparison
4	Reproducibility (seeds, README, runtime notes)	0.3	Full reproducibility: seeds, software versions, clear README
Total		**3.0**	

Rubric (condensed)

Criterion	Excellent	Satisfactory	Needs work
Model & pipeline	Correct CNN, tidy modules, verified shapes; training stable	Mostly correct with minor issues	Major architectural or training flaws
Augmentations	≥3 well-justified; ablation/impact quantified	≥2 used; limited analysis	Little/no augmentation or unjustified choices
Results	Strong/consistent metrics; curves clean; examples insightful	Reasonable metrics; curves present	Poor metrics; missing plots/examples
Comparison to Assignment 2	Rigorous, fair, and well-argued	Basic comparison present	Unfair or missing comparison
Reproducibility	Seeds, versions, clear README	Partial	Not reproducible

---

ADDITIONAL

Code scaffold (optional starter)

```
import torch, torch.nn as nn, torch.nn.functional as F
from torchvision import datasets, transforms

train_tf = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
])

test_tf = transforms.ToTensor()

class SmallCNN(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(64, 128, 3, padding=1), nn.ReLU()
        )
        self.classifier = nn.Linear(128, num_classes)  # via GAP
    def forward(self, x):
        x = self.features(x)               # [B,128,8,8]
        x = x.mean(dim=(2,3))             # GlobalAvgPool → [B,128]
        return self.classifier(x)
```

Tips

- Start small; confirm the loop by overfitting a tiny sample before scaling up.
- Monitor **learning rate** and **weight decay**; too high LR causes noisy curves.
- Keep augmentations **mild** to avoid underfitting; gradually add complexity.
- Use torch.cuda.amp.autocast() for mixed precision on GPU to speed up training.
- Log your runs (e.g., CSV/Matplotlib); screenshots are acceptable in the PDF.

Academic integrity & AI use

Using AI assistants (e.g., **Gemini in Colab** or others) is **encouraged** for boilerplate and debugging. You must:

- **Cite tools used** (one line in the report: tools + purpose).
- **Understand and explain** your model and results; we may ask brief viva questions.
- All submitted work must be **yours** (no copying classmates' code or reports).
